Kauno technologijos universitetas

Informatikos fakultetas

Statinė kodo analizė

T120B162 Programų sistemų testavimas

Trečias laboratorinis darbas

Autorius
Gustas Klevinskas

Akademinė grupė
IFF-8/7

Vadovas
Doc. Šarūnas Packevičius

Kaunas, 2021

Turinys

1. Rankinė kodo peržiūra

Kodo patikrinimui ranka pasirinkta remtis oficialiomis Microsoft rekomendacijomis C# programoms. Detaliau apie jas galima rasti internetiniame puslapyje https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions. Sekančiuose skyreliuose surašytos rekomendacijos iš šio puslapio ir pateikti atitinkami kodo pavyzdžiai iš projekto pademonstruojant, jog šios rekomendacijos sekamos.

Šios svetainės turinys turi daug programavimo srities žodžių, kurių lietuviškų atitikmenų sunku rasti, tad šioje ataskaitoje rekomendacijos surašytos originalia anglų kalba.

1.1. Naming conventions

1.1.1. Pascal case

Use pascal casing (“PascalCasing”) when naming a class, record, or struct.

```
public class PostController : ControllerBase {}
public class PostCreateModel {}
```

When naming public members of types, such as fields, properties, events, methods, and local functions, use pascal casing.

```
public class PostCreateModel {
    public string Title { get; set; } = string.Empty;
    public string Body { get; set; } = string.Empty;
}
```

1.1.2. Camel case

Use camel casing ("camelCasing") when naming private or internal fields, and prefix them with _.

```
public class PostController : ControllerBase {
    private readonly PostService _postService;
}

public class PostService {
    private readonly DataContext _dataContext;
    private readonly IMapper _mapper;
}
```

When working with static fields that are private or internal, use the s_ prefix and for thread static use t_.

Kode nenaudojami šie tipai.

1.2. Language guidelines

1.2.1. String data type

Use string interpolation to concatenate short strings.

```
$"{nameof(DateTo)} cannot be before {nameof(DateFrom)}"
```

To append strings in loops, especially when you're working with large amounts of text, use a String-Builder object.

Kode ranka nekuriami tokie eilutės tipai.

1.2.2. Implicitly typed local variables

Use implicit typing for local variables when the type of the variable is obvious from the right side of the assignment, or when the precise type is not important.

```
var post = mapper.Map<Post>(model);
```

```
var post = dataContext.Posts.Find(id);
```

1.2.3. try-catch and using statements in exception handling

Use a try-catch statement for most exception handling.

```
try {
    await next(context);
} catch (Exception error) {
    var response = context.Response;
    response.ContentType = "application/json";

    response.StatusCode = error switch {
        AppException => (int)HttpStatusCode.BadRequest,
        KeyNotFoundException => (int)HttpStatusCode.NotFound,
        _ => (int)HttpStatusCode.InternalServerError
    };
}
```

Simplify your code by using the C# using statement.

```
await using var appContext = Server.Services.GetRequiredService<DataContext>();
await appContext.Database.ExecuteSqlRawAsync($"UPDATE \"Users\" SET \"Role\" = {(int) Role.Admin} WHERE
\"Id\" = '{createdUser.Id}'");
```

1.2.4. “new” operator

Use one of the concise forms of object instantiation.

```
private readonly Lazy<IConfigurationProvider> _configProvider = new(CreateAutoMapperConfig, true);
```

Use object initializers to simplify object creation.

```
var postModel = new PostCreateModel {
    Title = "Test post",
    Description = "Test description",
    Body = "Test body",
    ImageUrl = "https://example.com/image",
    ImageLabel = "Example image"
};
```

2. Statinė kodo analizė

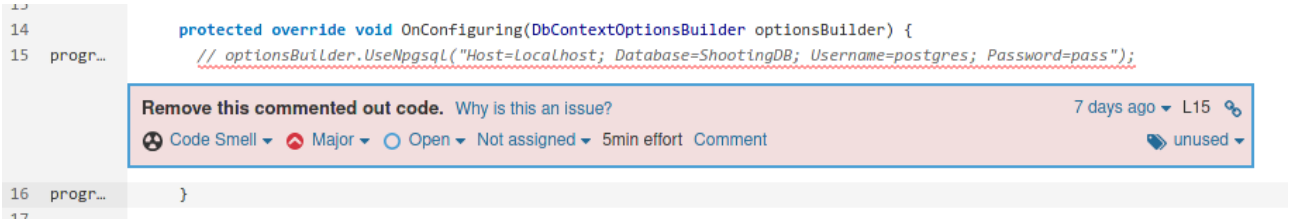
Statinei kodo analizei atlikti pasirinkta naudoti SonarQube. Jo rezultatai pateikti paveiksliukuose žemiau. Kadangi programuota naudojant modernius įrankius – Rider teksto redaktorių, ir sekta iškilusiais perspėjimais bei pranešimais, problematiškų vietų rasta labai nedaug. Tačiau net ir tos rastos 3 klaidos nėra labai teisingos.

1 pav. *Pirmoji rasta klaida*

Ši klasė atsakinga už darbą su C# išimtimis ir serverio HTTP atsakymais. Vietomis ši klasė leidžia išmesti išimtį (angl. *throw an exception*) vietoj konkrečių HTTP kodų ir žinučių grąžinimui kontroleriuose.

2 pav. *Antroji rasta klaida*

Anksčiau minėta klasė naudojama klaidoms tvarkyti iš autorizacijos bei autentifikacijos metodų. Ją galima būtų panaudoti ir įprastuose kontroleiuose, tačiau tuomet būtų prarandami tvarkingi HTTP atsakymai klaidų atveju. Grąžinant šias reikšmes iš kontrolerių, ASP.NET karkasas automatiškai uždeda papildomus laukus, kurie parinkti pagal RFC.



3 pav. *Trečia rasta klaida*

Čia yra iškarpa iš migracijų failo. Dar nepavykus rasti patogaus būdo tvarkyti duomenų bazės migracijas vietinėje darbinėje aplinkoje bei serveryje, iš kurio pasiekiamas šis API, pasirinkta prireikus atkomentuoti šią eilutę. Taip palengvinamas darbas lokaliai.

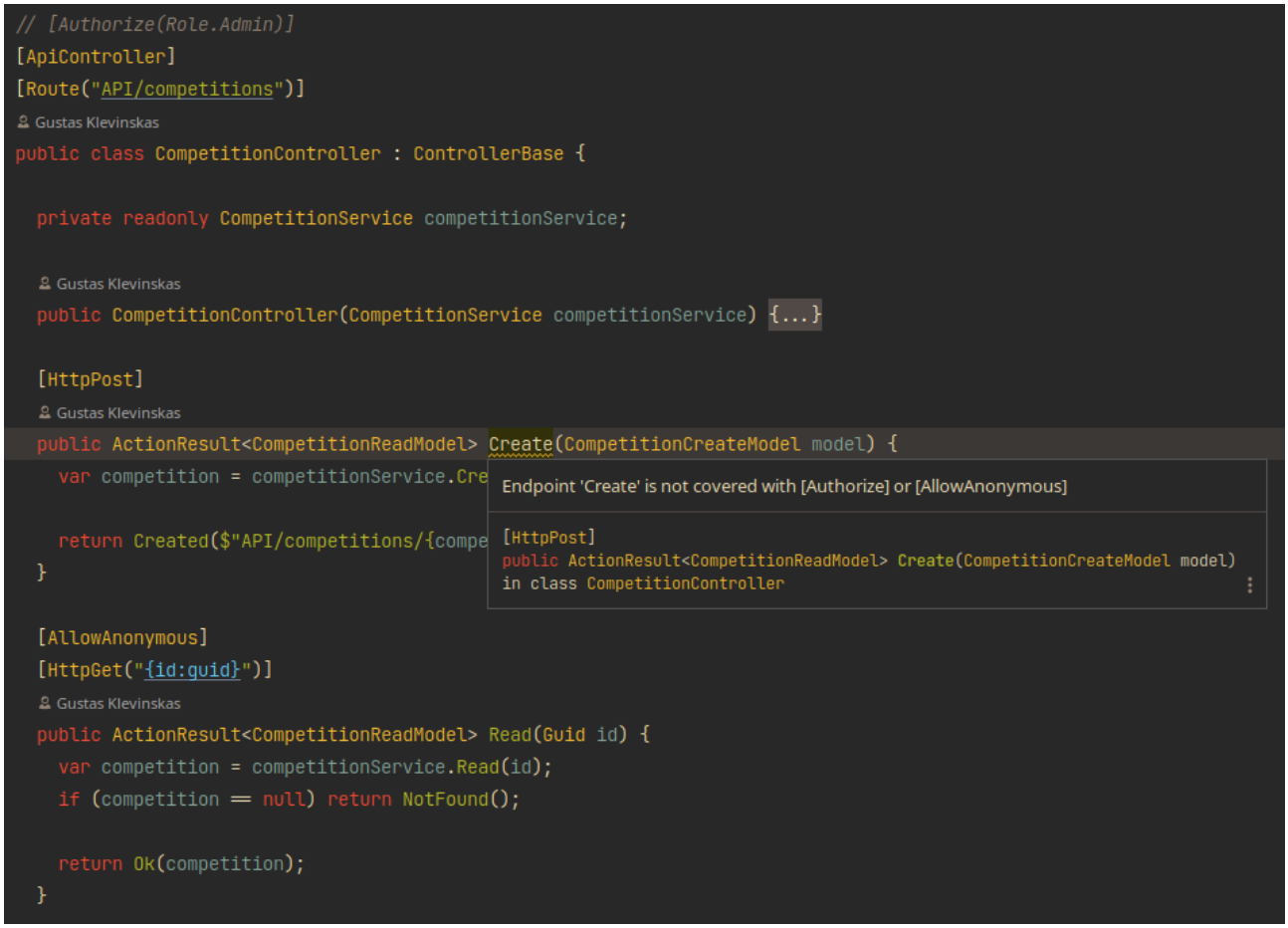
3. Savo sukurta analizės taisyklė

Su SonarQube pasirinkta atlikti statinę analizę, nes kodas ir taip buvo atsakingai rašomas naudojant standartinius Roslyn analizatorius ir norėta patikrinti ar atskiras įrankis ras kitokias klaidas bei rekomendacijas. Tačiau su SonarQube buvo gana sunku implementuoti savo taisyklę, tad šiam tikslui pasirinkta grįžti prie Roslyn – sukurtas asmeninis Roslyn analizatorius.

Taisyklė – metodai, priimantys HTTP užklausas, privalo turėti *Authorize* arba *AllowAnonymous* atributus. Jei pati kontrolerio klasė yra anotuota tokiu atributu, tuomet tai nėra būtina prie kiekvieno metodo. Nebent norima perrašyti klasės atributo reikšmę. Su šia taisykle tikėtasi išvengti situacijų, kai programuotojo užmiršta aiškiai nurodyti, ar naujas endpoint'as prieinamas viešai, ar tik tam tikrą rolę turintiems naudotojams.

Žemiau pateiktas pagrindinis kodo fragmentas, atsakingas už analizę.

```
    [DiagnosticAnalyzer(LanguageNames.CSharp)]
    public class L3Analyzer : DiagnosticAnalyzer
    {
        public const string DiagnosticId = "L3";

        private const string Title = "Enpoints should have an authorization attribute.";
        private const string MessageFormat = "Endpoint '{0}' is not covered with [Authorize] or
[AllowAnonymous]";
        private const string Description = "Endpoint is not covered with [Authorize] or [AllowAnonymous]";
        private const string Category = "Security";

        private static readonly DiagnosticDescriptor Rule = new DiagnosticDescriptor(DiagnosticId, Title,
MessageFormat, Category, DiagnosticSeverity.Warning, isEnabledByDefault: true, description: Description);

        public override ImmutableArray<DiagnosticDescriptor> SupportedDiagnostics { get { return
ImmutableArray.Create(Rule); } }

        public override void Initialize(AnalysisContext context)
        {
            context.ConfigureGeneratedCodeAnalysis(GeneratedCodeAnalysisFlags.None);
            context.EnableConcurrentExecution();
            context.RegisterSymbolAction(AnalyzeSymbol, SymbolKind.Method);
        }

        private static void AnalyzeSymbol(SymbolAnalysisContext context)
        {
            var parentClass = (ITypeSymbol) context.Symbol.ContainingType;

            if (parentClass == null) return;
            if (parentClass.BaseType.Name != "ControllerBase") return;

            var controllerAttributes = parentClass.GetAttributes().Select(x =>
x.AttributeClass.Name).ToList();
            if (controllerAttributes.Any(x => x.Contains("Authorize") || x.Contains("AllowAnonymous")))
return;

            var method = (IMethodSymbol) context.Symbol;
            var methodAttributes = method.GetAttributes().Select(x => x.AttributeClass.Name).ToList();

            if (!methodAttributes.Any(x => x.Contains("Http"))) return;
            if (methodAttributes.Any(x => x.Contains("Authorize") || x.Contains("AllowAnonymous"))) return;

            var diagnostic = Diagnostic.Create(Rule, context.Symbol.Locations[0], method.Name);
            context.ReportDiagnostic(diagnostic);
        }
    }
```



4 pav. *Perspėjimo pranešimo pavyzdys*

4 pav. pateiktas kodo redaktoriaus vaizdas, kai „Create“ metodas nebuvo paanotuotas nei pats, nei kontrolerio klasė, kurioje jis patalpintas. Tačiau „Read“ metodo nebraukia, nes jis pats paanotuotas *AllowAnonymous* atributu.

```
[Authorize(Role.Admin)]
[ApiController]
[Route("API/competitions")]
Gustas Klevinskas
public class CompetitionController : ControllerBase {

  private readonly CompetitionService competitionService;

  Gustas Klevinskas
  public CompetitionController(CompetitionService competitionService) {...}

  [HttpPost]
  Gustas Klevinskas
  public ActionResult<CompetitionReadModel> Create(CompetitionCreateModel model) {
    var competition = competitionService.Create(model);

    return Created($"API/competitions/{competition.Id}", competition);
  }

  [AllowAnonymous]
  [HttpGet("{id:guid}")]
  Gustas Klevinskas
  public ActionResult<CompetitionReadModel> Read(Guid id) {
    var competition = competitionService.Read(id);
    if (competition == null) return NotFound();

    return Ok(competition);
  }
```

5 pav. *Sutvarkytas kodas pagal pasiūlymą*

5 pav. rodomas sutvarkytas kodas – CompetitionController pažymėtas su *Authorize* atributu. Tuomet kodo redaktorius neberodo perspėjimo pranešimo.

4. Išvados

Pavyko sėkmingai išanalizuoti kodą ranka bei keliais skirtingais įrankiais: SonarQube ir Roslyn analizatoriais. Pamatyta, jog kodas rašomas tvarkingai, įrankiai didelių klaidų neaptiko. O kelios aptiktos klaidos tik dalinai naudingos – ateityje numatoma suvienodinti C# išimčių tvarkymą bei duomenų bazės migracijas.

Sukurtas naudinga analizatoriaus taisyklė, padėsianti palaikant HTTP metodų saugumą.